

Autor & Copyright: Dipl.-Ing. Harald Nahrstedt

Version: 2016 / 2019 / 2021 / 365

Erstellungsdatum: 01.12.2011

Überarbeitung: 01.12.2023

Beschreibung:

Den eigentlichen Funktionsumfang erhält Excel durch seine Objektbibliotheken. Einige dieser Bibliotheken (Libraries) sind bei der Standardinstallation bereits aktiv. Einige davon sind immer aktiv und können nicht deaktiviert werden. Andere müssen, wenn sie genutzt werden sollen, erst durch Verweise aktiviert werden. Neben diesen Standardbibliotheken gelten auch Mappen und Add-Ins als Bibliotheken, sofern sie Prozeduren und/oder Funktionen enthalten. Auch auf Mappen lassen sich Verweise einrichten mit dem Vorteil, dass deren öffentliche Variablen und Strukturen mitbenutzt werden können.

Anwendungs-Datei: AE-015\_Objektbindung.xlsm

## 1 Späte Bindung

Wenn Sie ein Worksheet-Objekt unter VBA nutzen möchten, so können sie dies in folgender Weise tun.

*Codeliste 1. Späte Bindung eines Worksheet-Objekts*

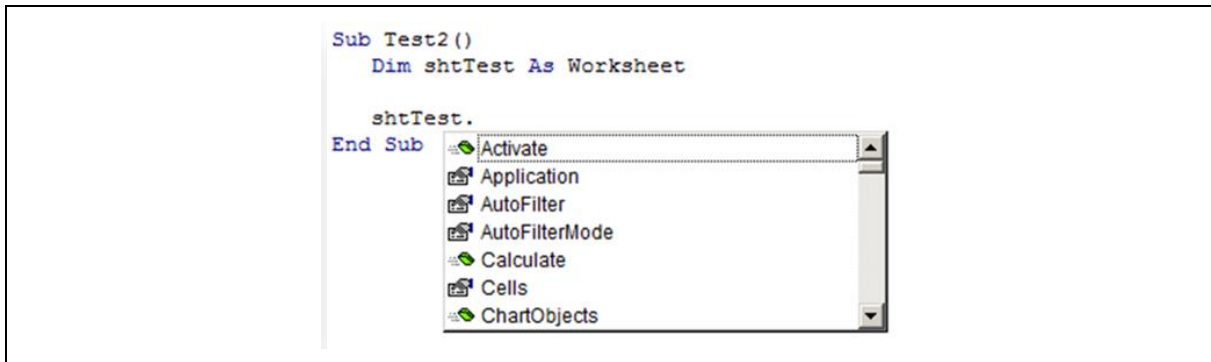
```
Sub Test()  
    Dim shtTest As Object  
    Dim sText As String  
  
    Set shtTest = ThisWorkbook.Worksheets("Tabelle1")  
    shtTest.Name = "TEST"  
End Sub
```

Zunächst definieren Sie einen Objekttyp (hier mit Namen shtTest). Die Präfix *sht* des Namens benutze ich für die Kennzeichnung, um welches Objekt es sich handelt (eben *sht* für Sheet). Erst mit der Initiierung des Objekts durch die Set Anweisung bekommt das Objekt *shtTest* die Attribute und Eigenschaften eines Sheets. Ein Attribut ist der Name des Scheets, der auf dem Registerblatt des Sheets steht. War er vorher *Tabelle1*, so ist er nach dem Prozedurlauf *TEST*. Welche Attribute ein Sheet besitzt, können Sie nur aus dem Objektkatalog der Entwicklungsumgebung erfahren.

Diese Form der Typzuweisung wird als *späte Bindung* (Late Binding) bezeichnet.

## 2 Frühe Bindung

Anders sieht es aus, wenn der Objekttyp bereits bei der Deklaration festgelegt wird (Bild 1).



*Bild 1. Frühe Bindung bietet die Anwendung der Intellisense-Funktion*

Bereits nach der Deklaration als Worksheet, bekommen Sie nach Eingabe des Objektnamens und dem Punkt, nach dem ein Unterobjekt, Attribute und Methoden angegeben werden können, eine Liste eingeblendet, die alle zulässigen Möglichkeiten enthält. Zugegeben, eine schöne Entwicklungshilfe, die als Intellisense der Entwicklungsumgebung bezeichnet wird. Lästiges Suchen und Probieren entfällt. Diese Methode bezeichnet man als *frühe Bindung* (Early Binding).

*Codeliste 2. Frühe Bindung eines Worksheet-Objekts*

```
Sub Test2()  
    Dim shtTest As Worksheet  
    Dim sText As String  
  
    Set shtTest = ThisWorkbook.Worksheets("Tabelle1")  
    shtTest.Name = "TEST"  
    Set shtTest = Nothing  
End Sub
```

## 3 Object Libraries

Voraussetzung ist allerdings, dass ein Verweis zur Objektbibliothek (Library) existiert. Welche Bibliotheken angebunden sind, finden Sie in der Entwicklungsumgebung unter dem Menü *Extras / Verweise* (Bild 2).

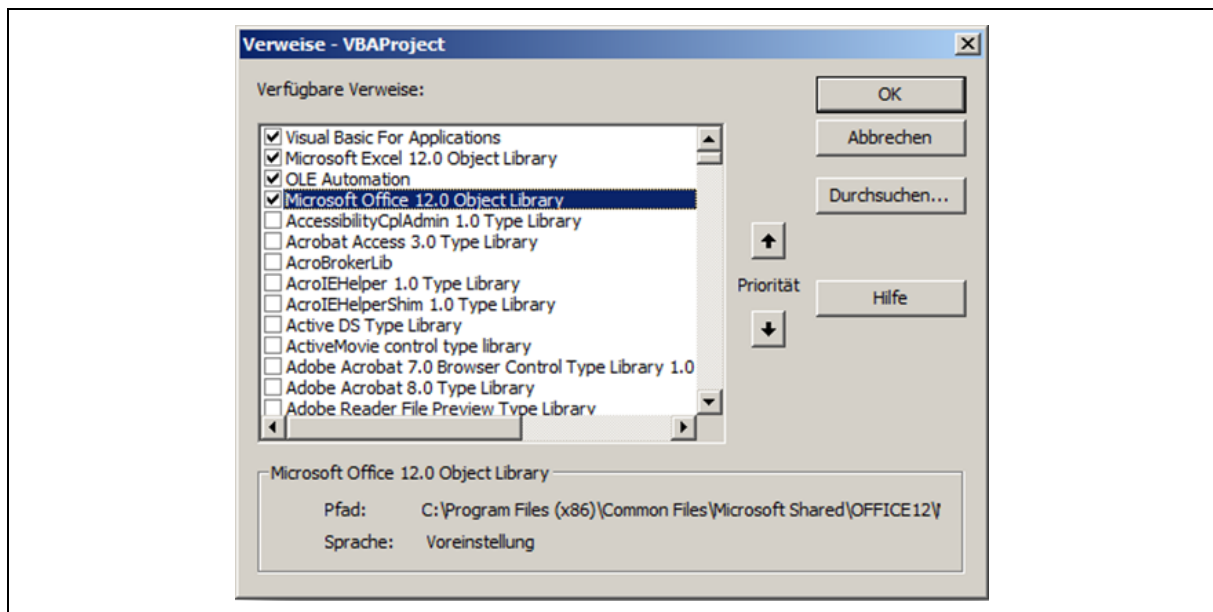


Bild 2. Verweise auf Objekt-Bibliotheken

Die Microsoft Excel Objekt-Bibliothek ist standardmäßig angebunden, so dass es hier keine Probleme gibt. Aber wenn Sie Elemente der Word-Bibliothek unter Excel nutzen möchten, müssen Sie diese unter Verweise mit angeben (Bild 3).

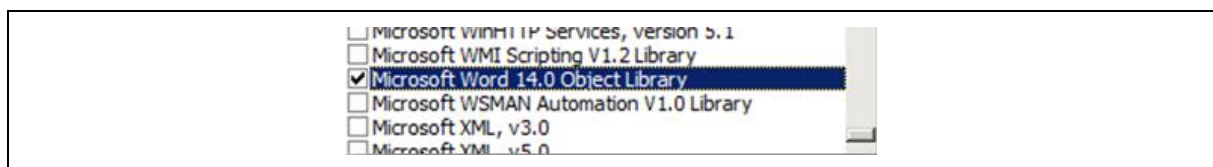


Bild 3. Verwaltung der Bibliotheks-Verweise

Danach können sie auch Word-Objekte und –Unterobjekte mit früher Bindung erstellen (Bild 4).

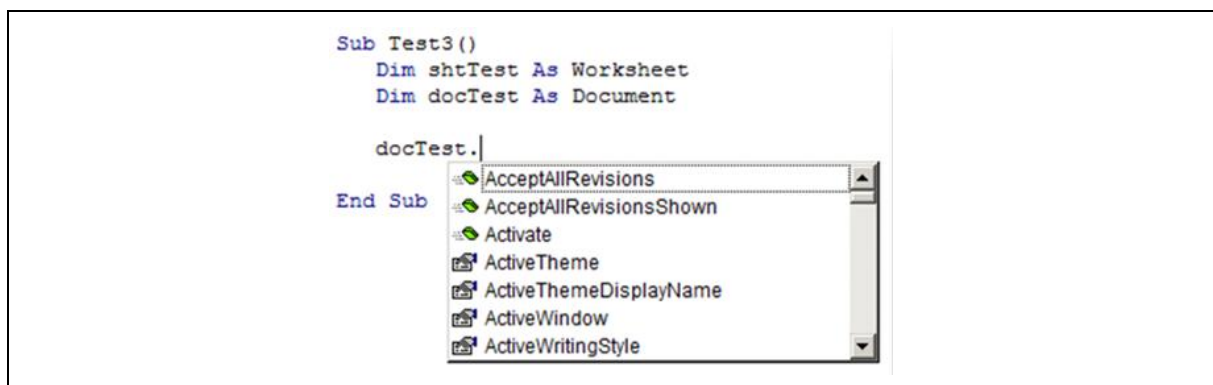


Bild 4. Objekte einer ausgewählten Bibliothek durch frühe Bindung einsetzen

#### 4 Verweise auf Objektbibliotheken lesen

Darin liegt aber auch ein Problem. Wird die Excelmappe auf einem anderen Rechner installiert, dann müssen dort ebenfalls die gleichen Verweise vorliegen. Oft entwickelt man Prozeduren mit der frühen Bindung und in der finalen Version bekommen dann die Objekte eine späte Bindung.

Doch VBA wäre nicht VBA, wenn es nicht dafür eine programmierbare Lösung gäbe. Die folgende Prozedur zeigt zunächst einmal alle vorhandenen Verweise im Direktfenster.

### Codeliste 3. Die Prozedur zeigt alle vorhandenen Verweise

```
Sub ShowReferences()  
    Dim objBook As Object  
    Dim refAll As Object  
    Dim refIst As Object  
  
    For Each objBook In Workbooks  
        Set refAll = objBook.VBProject.References  
        For Each refIst In refAll  
            Debug.Print objBook.Name, _  
                refIst.Name, _  
                refIst.Type, _  
                refIst.GUID, _  
                refIst.FullPath  
        Next  
    Next  
End Sub
```

Die damit ausgegebene GUID (global unique identifier) ist auf allen Rechnern weltweit gültig. Diese Methode ist sicherer, als mit den Installationspfaden zu arbeiten. Diese müssen nicht immer auf allen Rechnern stimmen.

Die folgende Prozedur prüft, ob alle Verweise in Ordnung sind.

### Codeliste 4. Die Prozedur prüft alle vorhandenen Verweise auf Gültigkeit

```
Sub VerifyReferences()  
    Dim objRef As Object  
  
    For Each objRef In ThisWorkbook.VBProject.References  
        If objRef.IsBroken Then _  
            MsgBox "Der Verweis auf " & objRef.Name & _  
                " ist defekt.", vbExclamation, "Achtung"  
        Next  
    End Sub
```

## 5 Verweise auf Objektbibliotheken installieren

Sollen Verweise auf Objektbibliotheken installiert werden, so ist zunächst zu prüfen, ob sie nicht bereits installiert sind. Sind sie gesetzt, dann sollte geprüft werden, ob sie in Ordnung sind. Sind sie nicht in Ordnung, dann müssen sie gelöscht und wieder neu gesetzt werden.

### Codeliste 5. Die Prozedur installiert Verweise unter Beachtung ihrer Gültigkeit

```
Private Sub InstallReferences()  
    Dim objRef As Object  
  
    Dim sRefGUID(2) As String  
    Dim iCount As Integer  
    'Office / MSO.DLL  
    sRefGUID(1) = "{2DF8D04C-5BFA-101B-BDE5-00AA0044DE52}"  
    'Word / MSWORD.OLB  
    sRefGUID(2) = "{00020905-0000-0000-C000-000000000046}"  
  
    'check bevor  
    For Each objRef In ThisWorkbook.VBProject.References  
        If objRef.IsBroken Then  
            objRef.Remove  
        Else  
            For iCount = 1 To 2  
                If objRef.GUID = sRefGUID(iCount) Then  
                    sRefGUID(iCount) = ""  
                End If  
            Next iCount  
        End If  
    Next  
  
    'set references  
    For iCount = 1 To 2  
        If Not sRefGUID(iCount) = "" Then  
            ThisWorkbook.VBProject.References.AddFromGuid _  
                GUID:=sRefGUID(iCount), Major:=2, Minor:=0  
        End If  
    Next iCount  
  
    'check after  
    For Each objRef In ThisWorkbook.VBProject.References
```

```

    If objRef.isbroken Then
        MsgBox "Der Verweis auf " & objRef.Name &
            " ist defekt.", vbExclamation, "Achtung"
    End If
Next

    ShowReferences
End Sub

```

## 6 Verweise auf Objektbibliotheken deinstallieren

So wie die Verweise installiert werden, lassen sie sich auch wieder deinstallieren. Dazu wird die Methode *Remove* verwendet.

*Codeliste 6. Die Prozedur deinstalliert Verweise*

```

Sub DeinstallReferences()
    Dim objRef      As Object

    Dim sRefGUID(2) As String
    Dim iCount      As Integer
    'Office / MSO.DLL
    sRefGUID(1) = "{2DF8D04C-5BFA-101B-BDE5-00AA0044DE52}"
    'Word / MSWORD.OLB
    sRefGUID(2) = "{00020905-0000-0000-C000-000000000046}"

    'check bevor
    On Error Resume Next
    For Each objRef In ThisWorkbook.VBProject.References
        For iCount = 1 To 2
            If objRef.GUID = sRefGUID(iCount) Then
                ThisWorkbook.VBProject.References.Remove objRef
            End If
        Next iCount
    Next
    ShowReferences
End Sub

```